2. The fast Fourier transform and fast multiplication

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

Lecture schedule

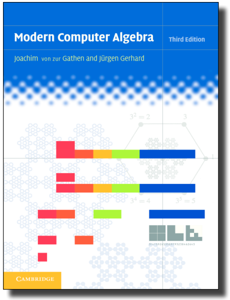
Tue 15 Jan:	1. Polynomials and integers
Tue 22 Jan:	2. The fast Fourier transform and fast multiplication
Tue 29 Jan:	3. Quotient and remainder
Tue 5 Feb:	4. Batch evaluation and interpolation
Tue 12 Feb:	5. Extended Euclidean algorithm and interpolation from erroneous data
Tue 19 Feb:	*Exam week — no lecture*
Tue 27 Feb:	6. Identity testing and probabilistically checkable proofs
Tue 5 Mar:	*Break — no lecture*
Tue 12 Mar:	7. Finite fields
Tue 19 Mar:	8. Factoring polynomials over finite fields
Tue 26 Mar:	9. Factoring integers

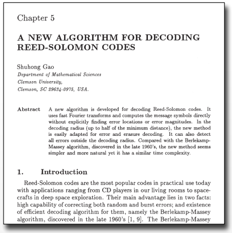
Recap of last week

- A boot camp of basic concepts and definitions in algebra
- Polynomials in one variable (univariate polynomials)
- Basic tasks and first algorithms for univariate polynomials
    - addition
    - multiplication
    - division (quotient and remainder)
    - evaluation
    - interpolation
    - greatest common divisor
- Evaluation–interpolation -duality of polynomials
- Analysis of the extended Euclidean algorithm via invariants

Goal: Near-linear-time toolbox for univariate polynomials

- Multiplication (this week)
- Division (quotient and remainder)
- Batch evaluation
- Interpolation
- Extended Euclidean algorithm (gcd)
- Interpolation from partly erroneous data




Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-0975, USA.

Abstract A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

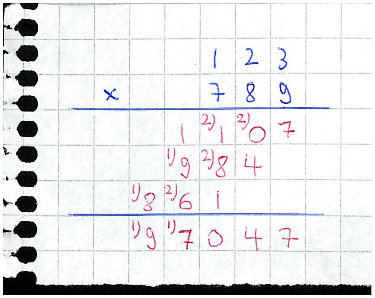
Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Further motivation for this week (1/2)

- The fast Fourier transform (FFT) is one of the most widely deployed and useful algorithms in all of computing
- Quick demo:
  FFT and fast polynomial multiplication (fast convolution) over $\mathbb{C}$ in signal processing
- In the exercises we will
  (a) derive an FFT over a ring R endowed with a primitive root of unity ω of order a power of 2, and
  (b) prove a version of the convolution theorem

Further motivation for this week (2/2)

- The classical integer multiplication algorithm has quadratic complexity
- But how to handle, say, gigabyte-size operands?
- We study the key ideas for FFT-based fast integer multiplication (exercise)
- See also:
  - `https://gmplib.org`
  - M. Fürer, Faster integer multiplication, *SIAM J. Comput.* 39 (2009) 979–1005 [9].
  - D. Harvey, J. van der Hoeven, G. Lecerf, Even faster integer multiplication, *J. Complexity* 36 (2016) 1–30 [13].

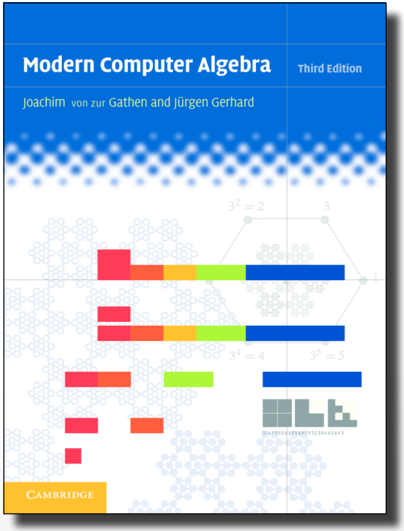


Key content for Lecture 2

- Evaluation–interpolation duality of polynomials
- Multiplication is a pointwise product in the dual
- Transforming between the primal and a (carefully chosen) dual —**roots of unity** and the **discrete Fourier transform** (DFT)
- The positional number system for integers
- Factoring a composite-order DFT to obtain a **fast Fourier transform** (FFT)
- Fast cyclic convolution (assuming a suitable root of unity exists)
- Fast negative-wrapping cyclic convolution

Fast multiplication

(von zur Gathen and Gerhard [11], Sections 8.2 and 8.3)



Coefficient and evaluation representations

- Let F be a field and let $\xi_0, \xi_1, \ldots, \xi_d \in F$ be distinct
- Let $a = \alpha_0 + \alpha_1 x + \ldots + \alpha_{d-1}x^{d-1} + \alpha_d x^d \in F[x]$ be a polynomial of degree at most d
- We can represent a as a list $(\alpha_0, \alpha_1, \ldots, \alpha_d) \in F^{d+1}$ of $d+1$ **coefficients**
- Alternatively, we can represent a as a list of $d+1$ **values** $(a(\xi_0), a(\xi_1), \ldots, a(\xi_d)) \in F^{d+1}$
- Indeed, we have

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix} = \begin{bmatrix} a(\xi_0) \\ a(\xi_1) \\ \vdots \\ a(\xi_d) \end{bmatrix}$$

  and the left-hand side Vandermonde matrix is invertible over F
  (recall exercise from last week)

Evaluation and interpolation

- To **evaluate** a polynomial $(\alpha_0, \alpha_1, \ldots, \alpha_d) \in F^{d+1}$ at distinct points $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix} = \begin{bmatrix} a(\xi_0) \\ a(\xi_1) \\ \vdots \\ a(\xi_d) \end{bmatrix}$$

- To **interpolate** the coefficients of a polynomial with values $(a(\xi_0), a(\xi_1), \ldots, a(\xi_d)) \in F^{d+1}$ at distinct $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the inverse of the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix}^{-1} \begin{bmatrix} a(\xi_0) \\ a(\xi_1) \\ \vdots \\ a(\xi_d) \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}$$

Example (evaluation)

- Let us evaluate $a = 1 + 2x + 3x^2 + 4x^3 + 5x^4 \in \mathbb{Z}_{13}$ at $\xi_0 = 0$, $\xi_1 = 1$, $\xi_2 = 2$, $\xi_3 = 3$, $\xi_4 = 4$ in $\mathbb{Z}_{13}$
- We have

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 \\ 1 & 3 & 9 & 1 & 3 \\ 1 & 4 & 3 & 12 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 12 \\ 1 \\ 7 \end{bmatrix}$$

  and hence $a(0) = 1$, $a(1) = 2$, $a(3) = 12$, $a(4) = 1$, $a(5) = 7$

Example (interpolation)

- Let us interpolate the coefficients of the unique polynomial $a \in \mathbb{Z}_{13}$ of degree at most 4 with values $a(\xi_0) = 1$, $a(\xi_1) = 2$, $a(\xi_2) = 12$, $a(\xi_3) = 1$, $a(\xi_4) = 7$ at $\xi_0 = 0$, $\xi_1 = 1$, $\xi_2 = 2$, $\xi_3 = 3$, $\xi_4 = 4$ in $\mathbb{Z}_{13}$
- We have

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 \\ 1 & 3 & 9 & 1 & 3 \\ 1 & 4 & 3 & 12 & 9 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 2 \\ 12 \\ 1 \\ 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 12 & 4 & 10 & 10 & 3 \\ 2 & 0 & 8 & 2 & 1 \\ 5 & 8 & 11 & 12 & 3 \\ 6 & 2 & 10 & 2 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 12 \\ 1 \\ 7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

  and hence $\alpha_0 = 1$, $\alpha_1 = 2$, $\alpha_2 = 3$, $\alpha_3 = 4$, $\alpha_4 = 5$
- The inverse of the Vandermonde matrix can be computed e.g. by Gaussian elimination or by using Lagrange polynomials (recall exercise from last week)

Evaluation–interpolation duality

- Evaluation–interpolation constitutes and example of two dual representations (coefficient and value representations of a polynomial)
- Both representations uniquely identify the object (the polynomial) under consideration
- Recall from the Introduction our learning objective:

  Making use of duality. Often a problem has a corresponding dual problem that is obtainable from the original (the primal) problem by means of an easy transformation. The primal and dual control each other, enabling and algorithm designer to use the interplay between the two representations

Evaluation–interpolation duality

- *Often a* **problem** *has a corresponding* **dual** *problem that is obtainable from the original (the* **primal***) problem by means of an easy* **transformation**
  - Polynomial multiplication (primal):
    Given coefficients of a and b as input, output coefficients of ab
  - Polynomial multiplication (dual):
    Given evaluations of a and b as input, output evaluations of ab
  - Transformation:
    Evaluation (primal $\rightarrow$ dual), interpolation (dual $\rightarrow$ primal)
- *The primal and dual control each other, enabling and algorithm designer to use the interplay between the two representations …*

Multiplication is easy in the dual

- Polynomial multiplication (dual):
  Given evaluations of a and b as input, output evaluations of ab
- Suppose $\deg a \leq n$ and $\deg b \leq m$
- Then $\deg ab \leq n + m$ and $n + m + 1$ evaluations of ab suffice to uniquely determine ab
- So suppose $\xi_0, \xi_1, \ldots, \xi_{n+m} \in F$ are distinct and we have the evaluations $a(\xi_0), a(\xi_1), \ldots, a(\xi_{n+m}) \in F$ and $b(\xi_0), b(\xi_1), \ldots, b(\xi_{n+m}) \in F$
- Then,
$$\begin{aligned} ab(\xi_0) &= a(\xi_0)b(\xi_0), \\ ab(\xi_1) &= a(\xi_1)b(\xi_1), \\ &\vdots \\ ab(\xi_{n+m}) &= a(\xi_{n+m})b(\xi_{n+m}) \in F \end{aligned}$$
- Thus, $O(n + m)$ multiplications suffice to determine ab, assuming we are in the dual

Multiplication in primal representation

- Polynomial multiplication (primal):
  Given coefficients of a and b as input, output coefficients of ab
- For $\deg a \leq d$ and $\deg b \leq d$, the classical algorithm uses $O(d^2)$ operations
- *The primal and dual control each other, enabling and algorithm designer to use the interplay between the two representations ...*
- Idea:
  1. Transform the inputs a and b into dual representation
  2. Multiply in the dual
  3. Transform the product ab back to primal representation
- Needed:
  Fast transformation between primal and dual representations

Fast evaluation and interpolation?

- To **evaluate** a polynomial $(\alpha_0, \alpha_1, \ldots, \alpha_d) \in F^{d+1}$ at distinct points $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix} = \begin{bmatrix} a(\xi_0) \\ a(\xi_1) \\ \vdots \\ a(\xi_d) \end{bmatrix}$$

- To **interpolate** the coefficients of a polynomial with values $(a(\xi_0), a(\xi_1), \ldots, a(\xi_d)) \in F^{d+1}$ at distinct $\xi_0, \xi_1, \ldots, \xi_d \in F$, we multiply from the left with the inverse of the Vandermonde matrix:

$$\begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^d \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^d \\ \vdots & \vdots & & \vdots \\ \xi_d^0 & \xi_d^1 & \cdots & \xi_d^d \end{bmatrix}^{-1} \begin{bmatrix} a(\xi_0) \\ a(\xi_1) \\ \vdots \\ a(\xi_d) \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}$$

Fast evaluation and interpolation?

- It is too expensive to construct the Vandermonde matrix (or its inverse) in explicit form
- Indeed, the matrix has $(d+1)^2$ elements in F, so working with the matrix in explicit form yields no better algorithms than classical multiplication in the primal representation
- For multiplication, both the input and the output use only $O(d)$ elements of F
- We have the freedom to choose any distinct $\xi_0, \xi_1, \ldots, \xi_d \in F$
- Perhaps a *good choice* enables evaluation and interpolation in $\tilde{O}(d)$ operations without constructing the Vandermonde matrix explicitly ...

Fast evaluation and interpolation?

- Idea:
  Choose
  $$\xi_0 = \omega^0, \quad \xi_1 = \omega^1, \quad \xi_2 = \omega^2, \quad \ldots, \quad \xi_d = \omega^d$$
  for a carefully chosen element $\omega \in F$ (whose existence depends on F)
- Intuition:
  With such a choice, the Vandermonde matrix should have a great deal of useful algebraic structure that maybe enables the use of, say, divide-and-conquer for matrix–vector multiplication, without even explicitly constructing the matrix ...

Key content (revisited)

- Evaluation–interpolation duality of polynomials
- Multiplication is a pointwise product in the dual
- Transforming between the primal and a (carefully chosen) dual —**roots of unity** and the **discrete Fourier transform** (DFT)
- The positional number system for integers
- Factoring a composite-order DFT to obtain a **fast Fourier transform** (FFT)
- Fast cyclic convolution (assuming a suitable root of unity exists)
- Fast negative-wrapping cyclic convolution

Roots of unity

- Let R be a ring (recall that we tacitly assume that R is commutative with $0 \neq 1$)
- For $n \in \mathbb{Z}_{\geq 1}$ and $\omega \in R$, we say that ω is a **root of unity** of **order** n in R if $\omega^n = 1$
- Or what is the same, ω is a root of the polynomial $z^n - 1 \in R[z]$

The discrete Fourier transform (DFT)

- Let ω be a root of unity of order n in R and let

$$f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_{n-1} x^{n-1} \in R[x]$$

- The n-**point discrete Fourier transform of** f **at** ω is the vector of evaluations

$$\mathrm{DFT}_\omega(f) = \hat{f} = (f(\omega^0), f(\omega^1), \ldots, f(\omega^{n-1})) \in R^n .$$

- Equivalently, we may view $\mathrm{DFT}_\omega : f \mapsto \hat{f}$ as the R-linear map that takes the vector of coefficients $f = (\varphi_0, \varphi_1, \ldots, \varphi_{n-1}) \in R^n$ to the vector $\hat{f} = (\hat{\varphi}_0, \hat{\varphi}_1, \ldots, \hat{\varphi}_{n-1}) \in R^n$ defined for all $i = 0, 1, \ldots, n-1$ by

$$\hat{\varphi}_i = \sum_{j=0}^{n-1} \varphi_j \omega^{ij}$$

Towards a first FFT: Splitting into even and odd parts

- Suppose that $n \in \mathbb{Z}_{\geq 2}$ is even and let $f = \sum_{i=0}^{n-1} \varphi_i x^i \in R[x]$
- Introduce the two polynomials

$$f_{\text{even}} = \sum_{i=0}^{n/2-1} \varphi_{2i} x^i \in R[x], \qquad f_{\text{odd}} = \sum_{i=0}^{n/2-1} \varphi_{2i+1} x^i \in R[x]$$

- We observe that

$$f(x) = f_{\text{even}}(x^2) + x \cdot f_{\text{odd}}(x^2)$$

- Here f has degree at most $n - 1$,
whereas f_{even} and f_{odd} have degree at most $n/2 - 1$

Towards a first FFT: Evaluating at a root of unity of order n

- Let $n \in \mathbb{Z}_{\geq 2}$ be even and $f = \sum_{i=0}^{n-1} \varphi_i x^i,\ f_{\text{even}} = \sum_{i=0}^{n/2-1} \varphi_{2i} x^i,\ f_{\text{odd}} = \sum_{i=0}^{n/2-1} \varphi_{2i+1} x^i$
- We recall that

$$f(x) = f_{\text{even}}(x^2) + x \cdot f_{\text{odd}}(x^2) \tag{4}$$

- Let $\omega \in R$ be a root of unity of order n; that is, $\omega^n = 1$
- We want to compute $f(\omega^0), f(\omega^1), \ldots, f(\omega^{n-1})$; that is, $\mathrm{DFT}_\omega(f)$
- From (4) and $\omega^n = 1$ we have that it suffices to first compute

$$\begin{aligned} f_{\text{even}}(\omega^0), f_{\text{even}}(\omega^2), \ldots, f_{\text{even}}(\omega^{2n-2}) \quad &\sim \quad f_{\text{even}}(\omega^0), f_{\text{even}}(\omega^2), \ldots, f_{\text{even}}(\omega^{n-2}) \\ f_{\text{odd}}(\omega^0), f_{\text{odd}}(\omega^2), \ldots, f_{\text{odd}}(\omega^{2n-2}) \quad &\sim \quad \underbrace{f_{\text{odd}}(\omega^0), f_{\text{odd}}(\omega^2), \ldots, f_{\text{odd}}(\omega^{n-2})}_{\text{That is, } \mathrm{DFT}_{\omega^2}(f_{\text{even}}) \text{ and } \mathrm{DFT}_{\omega^2}(f_{\text{odd}})} \end{aligned}$$

  and then do $O(n)$ arithmetic operations in R

A first FFT: Recursion and analysis

- We just saw that to compute the n-point $\mathrm{DFT}_\omega(f)$, it suffices to
  1. split f into the even part f_{even} and the odd part f_{odd}
  2. compute the $n/2$-point $\mathrm{DFT}_{\omega^2}(f_{\mathrm{even}})$,
  3. compute the $n/2$-point $\mathrm{DFT}_{\omega^2}(f_{\mathrm{odd}})$, and
  4. do $O(n)$ further arithmetic operations in R to recover $\mathrm{DFT}_\omega(f)$
- That is, the total number of arithmetic operations is $T(n) \leq 2 \cdot T(n/2) + O(n)$
- This is $T(n) = O(n \log_2 n)$ when $n = 2^k$ for $k \in \mathbb{Z}_{\geq 1}$ and we apply recursion

Primitive root of unity

- A root of unity $\omega \in R$ of order n is **primitive** if for any prime divisor t of n it holds that $\omega^{n/t} - 1$ is not a zero divisor in R
- *Examples:*
  - $\omega_n = \exp(2\pi i/n)$ is a primitive root of unity of order n in $\mathbb{C}$
  - 2 is a primitive root of unity of order 12 in $\mathbb{Z}_{13}$
  - For

k	29	71	75	95	108	123
ω	21	287	149	55	64	493

    we have that $p = k \cdot 2^{57} + 1$ is a prime and $\omega \in \mathbb{Z}_p$ is the least primitive root of unity of order 2^{57} in $\mathbb{Z}_p$

Properties of primitive roots of unity

Lemma 1

Let ω be a primitive root of unity of order n in R.
Then, for all integers s not divisible by n it holds that

(i) $\omega^s - 1$ *is not a zero divisor in* R*, and*

(ii) $\sum_{i=0}^{n-1} \omega^{is} = 0$

Lemma 2

Let ω be a primitive root of unity of order n in R.
Then, ω^a is a primitive root of unity of order $|n/a|$ in R for all divisors a of n

Proof of Lemma 1 I

- For any $\rho \in R$ and any $k \in \mathbb{Z}_{\geq 0}$, we have

$$(\rho - 1) \sum_{j=0}^{k-1} \rho^j = \rho^k - 1 \tag{5}$$

- Select any $s \in \mathbb{Z}$ that is not divisible by n. Since $\omega^n = 1$, we may assume $s = 1, 2, \ldots, n-1$
- Let $1 \leq g \leq s$ be the gcd of s and n with $us + vn = g$ for $u \in \mathbb{Z}_{\geq 0}$ and $v \in \mathbb{Z}_{\leq 0}$
- Since $s < n$, we can choose a prime divisor t of n so that g divides n/t
- Take $\rho = \omega^g$ and $k = n/(gt)$ in (5) to obtain that $\omega^g - 1$ divides $\omega^{n/t} - 1$. That is, $(\omega^g - 1)\gamma = \omega^{n/t} - 1$ for some $\gamma \in R$

Proof of Lemma 1 II

- Thus $\omega^g - 1$ cannot be a zero divisor since if it were, we could conclude that $0 = 0 \cdot \gamma = (\omega^g - 1)\beta\gamma = (\omega^{n/t} - 1)\beta$ for a nonzero $\beta \in R$ and hence $\omega^{n/t} - 1$ would be a zero divisor, a contradiction
- Take $\rho = \omega^s$ and $k = u$ in (5) to obtain that $\omega^s - 1$ divides $\omega^{us} - 1 = \omega^{us+vn} - 1 = \omega^g - 1$
- Thus, $\omega^s - 1$ cannot be a zero divisor since if it were, we could conclude that $\omega^g - 1$ is a zero divisor, a contradiction
- Take $\rho = \omega^s$ and $k = n$ in (5) to obtain $(\omega^s - 1)\sum_{i=0}^{n-1} \omega^{is} = \omega^{ns} - 1 = 1 - 1 = 0$
- Since $\omega^s - 1$ is not a zero divisor, we conclude that $\sum_{i=0}^{n-1} \omega^{is} = 0$ $\square$

Proof of Lemma 2

- Let a be a divisor of n
- For $|a| = n$ we observe that $\omega^n = \omega^{-n} = 1$ and hence $\omega^a = 1$ is trivially a primitive root of unity of order 1
- Suppose that $|a| < n$ and let t be any prime divisor of $|n/a| > 1$
- Then, $s = a|n/a|/t$ is not divisible by n and hence Lemma 1 implies that $\omega^s - 1 = (\omega^a)^{|n/a|/t} - 1$ is not a zero divisor
- Since t was arbitrary, ω^a is a primitive root of unity of order $|n/a|$ in R $\square$

The inverse discrete Fourier transform (inverse DFT)

Lemma 3

Suppose that n is a unit in R and let $\omega \in R$ be a primitive root of order n. Then,
$\mathrm{DFT}_{\omega}^{-1} = \frac{1}{n} \cdot \mathrm{DFT}_{\omega^{-1}}$

Proof.

It suffices to show that for all $k = 0, 1, \ldots, n-1$ we have

$$\varphi_k = \frac{1}{n} \sum_{i=0}^{n-1} \hat{\varphi}_i \omega^{-ik}$$

Take $s = j - k$ in Lemma 1 to conclude that

$$\frac{1}{n} \sum_{i=0}^{n-1} \hat{\varphi}_i \omega^{-ik} = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \varphi_j \omega^{ij} \omega^{-ik} = \sum_{j=0}^{n-1} \varphi_j \cdot \frac{1}{n} \sum_{i=0}^{n-1} \omega^{i(j-k)} = \varphi_k$$

$\square$

Beyond the first (radix-2) FFT

- The rest of the lecture goes beyond the first recursive derivation of a (radix-2) FFT where we assumed that
  1. the ring R has a (primitive) root of unity ω of order $n = 2^k$, and
  2. we are content with a recursive implementation
- The rest of the lecture contains more advanced material that shows how to
  1. unfold a (mixed-radix) recursion into a sequence of linear transformations suitable e.g. for parallel implementation; and
  2. work with rings that do not have a suitable root of unity
- But we will start with something well known and highly useful …

The positional number system (base B)

- Let $B \in \mathbb{Z}_{\geq 2}$
- Suppose that $\alpha \in \mathbb{Z}$ with $0 \leq \alpha \leq B^d - 1$ for some $d \in \mathbb{Z}_{\geq 0}$
- Then, there is a unique finite sequence

$$(\alpha_{d-1}, \alpha_{d-2}, \ldots, \alpha_1, \alpha_0) \in \mathbb{Z}_{\geq 0}^{d} \tag{6}$$

with $0 \leq \alpha_i \leq B - 1$ for all $i = 0, 1, \ldots, d - 1$ such that

$$\alpha = \sum_{i=0}^{d-1} \alpha_i B^i = \alpha_{d-1} B^{d-1} + \alpha_{d-2} B^{d-2} + \ldots + \alpha_2 B^2 + \alpha_1 B + \alpha_0 \tag{7}$$

- We say that the sequence (6) is the (d-digit) representation of the integer α in the positional number system with **base** B (or **radix** B)
- The elements α_i are the **digits** of α
- We say that α_{d-1} is the **most significant** digit and α_0 is the **least significant** digit

Example (base 10)

- Let us represent $123 \in \mathbb{Z}$ in base $B = 10$
- We have

$$123 = 1 \cdot 10^2 + 2 \cdot 10 + 3 \cdot 1$$

- Hence, the sequence $(1, 2, 3)$ represents 123 in base 10
- *Question/work point:*
  Given a representation in base B as input, how do you compute a representation in base C? Hint: quotient and remainder.
- (We will return to the work point in subsequent weeks.)

The positional number system (varying base)

- Let $B_{d-1}, B_{d-2}, \ldots, B_1, B_0 \in \mathbb{Z}_{\geq 2}$
- Suppose that $\alpha \in \mathbb{Z}$ with $0 \leq \alpha \leq B_{d-1}B_{d-2}\cdots B_1 B_0 - 1$
- Then, there is a unique finite sequence

$$(\alpha_{d-1}, \alpha_{d-2}, \ldots, \alpha_1, \alpha_0) \in \mathbb{Z}^d_{\geq 0} \tag{8}$$

with $0 \leq \alpha_i \leq B_i - 1$ for all $i = 0, 1, \ldots, d-1$ such that

$$\begin{aligned} \alpha &= \sum_{i=0}^{d-1} \alpha_i B_{i-1} B_{i-2} \cdots B_0 \\ &= \alpha_{d-1} B_{d-2} B_{d-3} \cdots B_0 + \ldots + \alpha_2 B_1 B_0 + \alpha_1 B_0 + \alpha_0 \end{aligned} \tag{9}$$

- We say that the sequence (8) is the representation of the integer α in the positional number system with (varying) **base** $(B_{d-2}, B_{d-1}, \ldots, B_1, B_0)$

Example (varying base)

- Let us represent $123 \in \mathbb{Z}$ in base $(9, 8, 7)$. We have

$$123 = 2 \cdot 8 \cdot 7 + 1 \cdot 7 + 4 \cdot 1$$

- Thus, the representation of 123 in base $(9, 8, 7)$ is $(2, 1, 4)$
- *Question/work point:*
  Given a representation in base $(B_{d-1}, B_{d-2}, \ldots, B_1, B_0)$ as input, how do you compute a representation in base $(C_{e-1}, C_{e-2}, \ldots, C_1, C_0)$? Hint: quotient and remainder.

Number systems

- Today:
  Positional number system for integers
- Next week:
  Radix-point representation and approximation of rational numbers
  (positional number system with integer and fractional parts separated by a radix point)
- Knuth [17, Chap. 4] gives an extensive treatment of number systems and algorithms for arithmetic
- Besides their use in arithmetic, varying-base representations of integers are encountered, for example, in linearization of array data in a memory hierarchy and in the design of vector-parallel algorithms (e.g. algorithms for GPUs), where an array of threads works with such linearizations

Factoring a composite-order DFT (1/3)

- Let ω be a primitive root of unity of composite order $n = st$ in R for integers $s, t \geq 2$
- We can view an index $k \in \{0, 1, \ldots, st-1\}$ as a varying-base integer $k = k_s t + k_t$ with $k_s \in \{0, 1, \ldots, s-1\}$ and $k_t \in \{0, 1, \ldots, t-1\}$
- That is, k_s and k_t are the digits of k in base (s, t) so that k_s is the most significant digit and k_t is the least significant digit
- Recall that for all $i = 0, 1, \ldots, st-1$ we have

$$\hat{\varphi}_i = \sum_{j=0}^{st-1} \varphi_j \omega^{ij}$$

- Let us expand the output index i in base (s, t) and the input index j in base (t, s)
- We have

$$\hat{\varphi}_{i_s t + i_t} = \sum_{j_s=0}^{s-1} \sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \omega^{(i_s t + i_t)(j_t s + j_s)}$$

Factoring a composite-order DFT (2/3)

- Expand and use the fact that $\omega^{st} = 1$ to obtain

$$\begin{aligned}
\hat{\varphi}_{i_s t + i_t} &= \sum_{j_s=0}^{s-1} \sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \omega^{(i_s t + i_t)(j_t s + j_s)} \\
&= \sum_{j_s=0}^{s-1} \sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \omega^{i_s j_t s t + i_s j_s t + i_t j_t s + i_t j_s} \\
&= \sum_{j_s=0}^{s-1} \sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \omega^{i_s j_s t} \omega^{i_t j_t s} \omega^{i_t j_s} \\
&= \underbrace{\sum_{j_s=0}^{s-1} \left(\omega^{t}\right)^{i_s j_s} \underbrace{\omega^{i_t j_s} \underbrace{\sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \left(\omega^{s}\right)^{i_t j_t}}_{\text{(i)}}}_{\text{(ii)}}}_{\text{(iii)}} \, .
\end{aligned}$$

Factoring a composite-order DFT (3/3)

- Let us study (i), (ii), and (iii) as the indices i_s, i_t, j_s, j_t range over their domains:

$$\hat{\varphi}_{i_s t + i_t} = \underbrace{\sum_{j_s=0}^{s-1} \left(\omega^t\right)^{i_s j_s} \underbrace{\omega^{i_t j_s} \underbrace{\sum_{j_t=0}^{t-1} \varphi_{j_t s + j_s} \left(\omega^s\right)^{i_t j_t}}_{\text{(i)}}}_{\text{(ii)}}}_{\text{(iii)}}$$

- Part (i) takes the $t \times s$ input f and outputs the $t \times s$ array obtained by taking the t-point discrete Fourier transform at ω^s for each of the s columns of f
- Part (ii) multiplies the resulting $t \times s$ array entrywise (Hadamard product) with the $t \times s$ Vandermonde matrix with entries $\omega^{i_t j_s}$
- Part (iii) takes as input the $t \times s$ array output by (ii) and outputs the $t \times s$ array obtained by taking the s-point discrete Fourier transform at ω^t for each of the t rows of the array
- Finally, transpose the $t \times s$ array to obtain the $s \times t$ output $\hat{f}$

Fast Fourier transform (FFT)

- Idea:
  Apply the previous factorization recursively for smooth n
- For example, suppose that $n = 2^k$ for $k \in \mathbb{Z}_{\geq 1}$
- Take $s = 2^{k-1}$ and $t = 2$
- Compute Parts (i) and (ii) explicitly, and apply the factorization recursively in Part (iii)
- Thus, the DFT at a primitive root of unity ω of order 2^k in R can be computed in $T(2^k) \leq 2T(2^{k-1}) + O(2^k)$ operations in R (exercise)
- In particular, $T(n) = O(n \log n)$

Factors in an FFT (1/4)

- Let us now look at a possible implementation of Parts (i), (ii), and (iii) in more detail
- Let ω be a primitive root of order $n \in \mathbb{Z}_{\geq 1}$ in R and let $n = pqr$ for $p, q, r \in \mathbb{Z}_{\geq 1}$
- We will study two types of transformations that take as input an array $a \in R^n$ and produce as output an array $b \in R^n$
- We assume that the entries $a[i] \in R$ of an array are indexed with $i = 0, 1, \ldots, n-1$
- Let $w \in R^n$ be an array with $w[i] = \omega^i$ for all $i = 0, 1, \ldots, n-1$
 (in an implementation this array can be precomputed with $O(n)$ operations in R)

Factors in an FFT (2/4)

- The first transformation $\Phi_{(p,q,r)} : R^n \to R^n$ sets
$$b[iqr + jr + k] = \sum_{\ell=0}^{q-1} w[(j\ell pr) \text{ rem } n]a[iqr + \ell r + k] \tag{10}$$
for all $i \in \{0, 1, \ldots, p-1\}$, $j \in \{0, 1, \ldots, q-1\}$, $k \in \{0, 1, \ldots, r-1\}$
- Observe that the transformation relies on integers in base (p, q, r) for indexing the input a and the output b
- Also observe that the transformation implements pr disjoint copies of a q-point DFT, using in total $O(pq^2r) = O(nq)$ operations in R
- This transformation can be used to implement Parts (i) and (iii)

Factors in an FFT (3/4)

- The second transformation $\Theta_{(p,q,r)} : R^n \to R^n$ sets
  $$b[iqr + jr + k] = w[jkp]a[iqr + jr + k] \tag{11}$$
  for all $i \in \{0, 1, \ldots, p-1\}$, $j \in \{0, 1, \ldots, q-1\}$, $k \in \{0, 1, \ldots, r-1\}$
- Again we observe that we work in base (p, q, r)
- This transformation runs in $O(pqr) = O(n)$ operations in R
- This transformation can be used to implement Part (ii)

Factors in an FFT (4/4)

- A naïve implementation of Parts (iii), (ii), and (i) would now implement an n-point DFT for $n = st$ and input $f \in R^n$ as a sequence of three transformations, read from right to left, $\Phi_{(1,t,s)}\Theta_{(1,s,t)}\Phi_{(1,s,t)}(f)$
- This must be followed by transposition of the resulting array from $t \times s$ to $s \times t$ to obtain the output $\hat{f}$ (*Why?*)
- However, this does not yet reduce the number of operations to $O(n \log n)$ (*Why?*)
- In an implementation with $n = 2^k$, one can fix $q = 2$ and proceed with a sequence of $2k - 1$ transformations, with $p = 2^j$ and $r = 2^{k-1-j}$ for $j = 0, 1, \ldots k - 1$ (completing the details are an exercise), followed by final permutation of the resulting array

Remarks

- The previous example gave one possibility to implement an FFT
- In general the term “fast Fourier transform” refers to a family of algorithms that rely on factoring an $n \times n$ Vandermonde matrix $\Omega = (\omega^{ij} : i, j = 0, 1, \ldots, n-1)$ for a composite n into a sequence of simpler (sparse) matrices such that matrix–vector multiplication with each matrix in the sequence is cheap to execute
- For example, you may want to view the transformations (10) and (11) as obtaining the vector b by multiplying a matrix with the vector a
- Van Loan [27] gives an extensive treatment of computational frameworks for the FFT

Key content (revisited)

- Evaluation–interpolation duality of polynomials
- Multiplication is a pointwise product in the dual
- Transforming between the primal and a (carefully chosen) dual —**roots of unity** and the **discrete Fourier transform** (DFT)
- The positional number system for integers
- Factoring a composite-order DFT to obtain a **fast Fourier transform** (FFT)
- Fast cyclic convolution (assuming a suitable root of unity exists)
- Fast negative-wrapping cyclic convolution

Ideal, principal ideal

- Let R be a ring
- A nonempty subset I of R is an **ideal** if
  1. for all $a, b \in I$ we have $a + b \in I$, and
  2. for all $a \in I$ and $r \in R$ we have $ar \in I$
- *Examples:*
  - For any $a \in R$ we have that $\langle a \rangle = aR = \{ar : r \in R\}$ is an ideal;
    we say that $\langle a \rangle$ is the **principal ideal** generated by $a \in R$
  - For any $n \in \mathbb{Z}$, we observe that $n\mathbb{Z} = \{\ldots, -2n, -n, 0, n, 2n, \ldots\}$ is a (principal) ideal of $\mathbb{Z}$

Congruence modulo an ideal,   residue class

- Let I be an ideal of R
- We say that $r, s \in R$ are **congruent modulo** I and write $r \equiv s \pmod{I}$ if $r - s \in I$
- For $r \in R$ we say that the set $r + I = \{r + a : a \in I\}$ is a **residue class modulo** I
- For all $r, s \in R$ we have

$$r + I = s + I \quad \Leftrightarrow \quad r - s \in I \quad \Leftrightarrow \quad r \equiv s \pmod{I}$$

Residue class ring (factor ring)

- Let I be an ideal of R
- The set $R/I = \{r + I : r \in R\}$ of all residue classes modulo I is a ring (the **factor ring** or **residue class ring** of R modulo I) if we define the ring operations for all $r, s \in R$ by

  $$(r + I) + (s + I) = (r + s) + I$$

  and

  $$(r + I)(s + I) = (rs) + I$$

- Observe in particular that the aforementioned operations are well-defined in the sense that they do not depend on the choices of representatives r, s for residue classes modulo I (exercise)
- *Example*:
  For $R = \mathbb{Z}$ and $I = n\mathbb{Z}$ with $n \in \mathbb{Z}_{\geq 1}$, we have $R/I = \mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}_n$

Example: Cyclic convolution

- Let R be a ring and let $n \in \mathbb{Z}_{\geq 1}$
- Consider the factor ring $R[x]/\langle x^n - 1\rangle$
- We may view the elements of $R[x]/\langle x^n - 1\rangle$ as polynomials of degree at most $n - 1$ in $R[x]$
- Addition and multiplication in $R[x]/\langle x^n - 1\rangle$ are as in $R[x]$, with the exception that after multiplication we simplify the result with the substitution $x^n = 1$

Example: Cyclic convolution

- Suppose that $n = 8$ and that $R = \mathbb{Z}_{17}$
- Let us multiply the following two polynomials in $R[x]/\langle x^n - 1\rangle$

$$f = 1 + 8x + 13x^2 + 16x^3 + 15x^4 + 6x^5 + 7x^6 + 10x^7$$
$$g = 4 + 3x + 16x^2 + 7x^3 + 6x^4 + 11x^5 + 9x^6 + 15x^7$$

- In $R[x]$, the product is

$$fg = 4 + x + 7x^2 + 4x^4 + 16x^5 + 12x^6 + 10x^7 + 7x^8 + x^9 + 9x^{10} + 8x^{11} + 8x^{12} + 8x^{13} + 14x^{14}$$

- In $R[x]/\langle x^n - 1\rangle$, we can first compute fg in $R[x]$ as above, reduce the result with the substitution $x^n = 1$, and then simplify to obtain the result in $R[x]/\langle x^n - 1\rangle$ (or, what is the same, first multiply in $R[x]$ and take the remainder in the division with $x^n - 1$):

$$\begin{aligned} fg &= 4 + x + 7x^2 + 4x^4 + 16x^5 + 12x^6 + 10x^7 + 7 + x + 9x^2 + 8x^3 + 8x^4 + 8x^5 + 14x^6 \\ &= 11 + 2x + 16x^2 + 8x^3 + 12x^4 + 7x^5 + 9x^6 + 10x^7 \end{aligned}$$

Cyclic convolution via the DFT

- Let R be a ring, let $n \in \mathbb{Z}_{\geq 1}$, and let $\omega \in R$ be a primitive root of unity of order n
- For two vectors $a = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) \in R^n$ and $b = (\beta_0, \beta_1, \ldots, \beta_{n-1}) \in R^n$, let us write $a \cdot b$ for the pointwise product $a \cdot b = (\alpha_0\beta_0, \alpha_1\beta_1, \ldots, \alpha_{n-1}\beta_{n-1}) \in R^n$

Theorem 4 (Convolution Theorem)

For all $f, g \in R[x]/\langle x^n - 1\rangle$ *we have* $\mathrm{DFT}_\omega(fg) = \mathrm{DFT}_\omega(f) \cdot \mathrm{DFT}_\omega(g)$

Proof.

Exercise. □

- Furthermore, if n is a unit in R, we have $fg = \frac{1}{n}\,\mathrm{DFT}_{\omega^{-1}}(\mathrm{DFT}_\omega(f) \cdot \mathrm{DFT}_\omega(g))$
- This enables fast algorithms for computing fg assuming (i) R admits a suitable primitive root of unity, (ii) n is a unit in R, and (iii) n is divisible enough to enable an FFT by divide and conquer

Example: Cyclic convolution via the DFT (1/2)

- Suppose that $n = 8$ and that $R = \mathbb{Z}_{17}$
- We observe that $\omega = 2$ is a primitive root of unity of order $n = 8$ in $R = \mathbb{Z}_{17}$; indeed,

$$\omega^0 = 1,\ \omega^1 = 2,\ \omega^2 = 4,\ \omega^3 = 8,\ \omega^4 = 16,\ \omega^5 = 15,\ \omega^6 = 13,\ \omega^7 = 9,\ \omega^8 = 1$$

- We also observe that $2^{-1} = 9 \in R = \mathbb{Z}_{17}$ and hence $8^{-1} = 2^{-3} = 15 \in R = \mathbb{Z}_{17}$
- Using the DFT, let us multiply the following two polynomials in $R[x]/\langle x^n - 1\rangle$

$$f = 1 + 8x + 13x^2 + 16x^3 + 15x^4 + 6x^5 + 7x^6 + 10x^7$$
$$g = 4 + 3x + 16x^2 + 7x^3 + 6x^4 + 11x^5 + 9x^6 + 15x^7$$

- First we compute the n-point DFTs of f and g at ω to obtain

$$\mathrm{DFT}_\omega(f) = (8, 11, 16, 7, 13, 9, 10, 2)$$
$$\mathrm{DFT}_\omega(g) = (3, 14, 4, 9, 16, 4, 0, 16)$$

Example: Cyclic convolution via the DFT (2/2)

- Next we take the pointwise product of the DFTs

$$\mathrm{DFT}_\omega(f) = (8, 11, 16, 7, 13, 9, 10, 2) \in R^n$$

$$\mathrm{DFT}_\omega(g) = (3, 14, 4, 9, 16, 4, 0, 16) \in R^n$$

  to obtain

$$\mathrm{DFT}_\omega(f) \cdot \mathrm{DFT}_\omega(g) = (7, 1, 13, 12, 4, 2, 0, 15) \in R^n$$

- Finally take the inverse n-point DFT to obtain the result

$$\frac{1}{n}\mathrm{DFT}_{\omega^{-1}}(\mathrm{DFT}_\omega(f) \cdot \mathrm{DFT}_\omega(g)) = (11, 2, 16, 8, 12, 7, 9, 10) \in R^n$$

  or what is the same as a polynomial

$$fg = 11 + 2x + 16x^2 + 8x^3 + 12x^4 + 7x^5 + 9x^6 + 10x^7$$

Remarks (1/2)

- Cyclic convolution via the DFT (when implemented with FFTs) can be used to multiply polynomials in $R[x]$ fast
- Indeed, simply choose a large enough n so that the degree of the product of the two polynomials is less than n
- In this situation we can multiply two polynomials using $O(n \log n)$ operations in R; contrast this with the classical $O(n^2)$ operations
- Caveat:
  This approach needs (i) that R is endowed with a primitive root of unity ω of order n and (ii) that n is a unit in R
- *So what to do when R does not meet (i) and (ii) ?*

Remarks (2/2)

- Next we will look at a multiplication algorithm, *Schönhage–Strassen multiplication* [24], that needs very light assumptions about the coefficient ring
- Our present exposition roughly follows the exposition of a polynomial version of the Schönhage–Strassen algorithm in von zur Gathen and Gerhard [11, Section 8.3]
- For convenience in what follows, let us write S instead of R for the coefficient ring, and y instead of x for the polynomial indeterminate
- Rather than relying on cyclic convolution, the algorithm will rely on the following notion of *negative-wrapping* cyclic convolution ...

Example: Negative-wrapping cyclic convolution

- Let S be a ring and let $n \in \mathbb{Z}_{\geq 1}$
- Consider the factor ring $S[y]/\langle y^n + 1\rangle$
- We may view the elements of $S[y]/\langle y^n + 1\rangle$ as polynomials of degree at most $n - 1$ in $S[y]$
- Addition and multiplication in $S[y]/\langle y^n + 1\rangle$ are as in $S[y]$, with the exception that after multiplication we simplify the result with the substitution $y^n = -1$

Example: Negative-wrapping cyclic convolution

- Suppose that $n = 8$ and that $S = \mathbb{Z}_5$
- Let us multiply the following two polynomials in $S[y]/\langle y^n + 1\rangle$

$$\begin{aligned} f &= 1 + 2y + 2y^2 + 4y^3 + 3y^4 + 4y^5 + 2y^6 + 3y^7 \\ g &= 3 + 2y + 4y^2 + y^4 + 4y^5 + y^6 + 2y^7 \end{aligned}$$

- In $S[y]$, the product is

$$fg = 3 + 3y + 4y^2 + 4y^3 + y^4 + 2y^6 + 4y^8 + y^9 + 4y^{10} + y^{11} + 2y^{12} + 2y^{13} + y^{14}$$

- In $S[y]/\langle y^n + 1\rangle$, we can first compute fg in $S[y]$ as above, reduce the result with the substitution $y^n = -1$, and then simplify to obtain the result in $S[y]/\langle y^n + 1\rangle$ (or, what is the same, first multiply in $S[y]$ and take the remainder in the division with $y^n + 1$):

$$\begin{aligned} fg &= 3 + 3y + 4y^2 + 4y^3 + y^4 + 2y^6 - 4 - y - 4y^2 - y^3 - 2y^4 - 2y^5 - y^6 \\ &= 4 + 2y + 3y^3 + 4y^4 + 3y^5 + y^6 \end{aligned}$$

Schönhage–Strassen multiplication (1/7)

- Let S be a ring
- Suppose that 2 is a unit in S (this is the only assumption we make about S)
- Let $n = 2^k$ for some $k \in \mathbb{Z}_{\geq 3}$
- Let $f, g \in S[y]/\langle y^n + 1\rangle$ be given as input
- We want to compute the product $fg \in S[y]/\langle y^n + 1\rangle$
- With foresight, let $m = 2^{\lfloor k/2 \rfloor}$ and $t = 2^{\lceil k/2 \rceil}$; in particular, we have $n = mt$ and $m \leq t \leq 2m$
- The key idea is to reduce one multiplication in $S[y]/\langle y^{mt} + 1\rangle$ into t multiplications in $S[y]/\langle y^{2m} + 1\rangle$ and then apply recursion

Running example (1/8)

- It will be convenient to illustrate the algorithm design with a running example
- Let us work with $S = \mathbb{Z}_5$; in particular we observe that 2 is a unit with inverse $2^{-1} = 3 \in \mathbb{Z}_5$
- Suppose that $n = 8$ and that our given input in $S[y]/\langle y^n + 1\rangle$ is

$$\begin{aligned} f &= 1 + 2y + 2y^2 + 4y^3 + 3y^4 + 4y^5 + 2y^6 + 3y^7 \\ g &= 3 + 2y + 4y^2 + y^4 + 4y^5 + y^6 + 2y^7 \end{aligned}$$

- We need to produce the output

$$fg = 4 + 2y + 3y^3 + 4y^4 + 3y^5 + y^6$$

- Since $n = 8$, we have $m = 2$ and $t = 4$

Schönhage–Strassen multiplication (2/7)

- Let us introduce a new indeterminate x and transform f and g so that every monomial y^k is replaced with $x^q y^r$ where q and r are the unique nonnegative integers with $k = qm + r$ and $0 \leq r < m$
- Let us write F and G for the resulting two-variable polynomials in $S[x, y]$
- Let $Q, H \in S[x, y]$ be the unique polynomials such that
$$FG = (x^t + 1)Q + H \tag{12}$$
and H has x-degree at most $t - 1$
- We observe that Q, H above exist by polynomial division (e.g. recall Lecture 1) since the leading coefficient of $x^t + 1$ is a unit in $S[y]$ with $(S[y])[x] \cong S[x, y]$
- (It should be noted that the actual algorithm never constructs Q in explicit form, here we merely use polynomial division to conclude that Q *exists*.)

Running example (2/8)

- Continuing the running example, we have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$ and the inputs

$$\begin{aligned} f &= 1 + 2y + 2y^2 + 4y^3 + 3y^4 + 4y^5 + 2y^6 + 3y^7 \\ g &= 3 + 2y + 4y^2 + \phantom{4y^3 +{}} y^4 + 4y^5 + y^6 + 2y^7 \end{aligned}$$

- Substituting $y^m = x$ to f and g, the polynomials F and G in $S[x, y]$ are

$$\begin{aligned} F &= 1 + 2y + (2 + 4y)x + (3 + 4y)x^2 + (2 + 3y)x^3 \\ G &= 3 + 2y + 4x + (1 + 4y)x^2 + (1 + 2y)x^3 \end{aligned}$$

- For illustration, let us also display the polynomials FG, Q, and H, but also observe that FG and Q are not computed by the algorithm, and the polynomial H will be obtained only later

$$\begin{aligned} FG &= 3 + 3y + 4y^2 + (4y + 3y^2)x + (3 + y^2)x^2 + (1 + y^2)x^3 + (3 + y + 4y^2)x^4 + yx^5 + (2 + 2y + y^2)x^6 \\ Q &= 3 + y + 4y^2 + yx + (2 + 2y + y^2)x^2 \\ H &= 2y + (3y + 3y^2)x + (1 + 3y)x^2 + (1 + y^2)x^3 \end{aligned}$$

Schönhage–Strassen multiplication (3/7)

- Substitute $x = y^m$ to both sides of (12) to conclude that

  $$F(y^m, y)G(y^m, y) = (y^{mt} + 1)Q(y^m, y) + H(y^m, y)$$

  implying

  $$F(y^m, y)G(y^m, y) \equiv H(y^m, y) \pmod{y^{mt} + 1}$$

- Since $f = F(y^m, y)$ and $g = G(y^m, y)$, we conclude that it suffices to compute $H(y^m, y)$ to determine the product fg in $S[y]/\langle y^{mt+1}\rangle$
- Indeed, $H(y^m, y)$ is a polynomial in y with degree less than $2mt$, which is easily reduced with the substitution $y^{mt} = -1$ to yield the result fg
- We observe that (12) implies $FG \equiv H \pmod{x^t + 1}$, so our goal in what follows will be to multiply given F and G modulo $x^t + 1$

Running example (3/8)

- Continuing the running example, we have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$ and

$$\begin{aligned} F &= 1 + 2y + (2+4y)x + (3+4y)x^2 + (2+3y)x^3 \\ G &= 3 + 2y + 4x + (1+4y)x^2 + (1+2y)x^3 \end{aligned}$$

- Let us also recall that

$$H = 2y + (3y + 3y^2)x + (1+3y)x^2 + (1+y^2)x^3$$

- Thus, substituting $y^m = x$ into H, we obtain

$$H(y^m, y) = 2y + 3y^3 + 4y^4 + 3y^5 + y^6 + y^8$$

- Substituting $y^{mt} = -1$ into $H(y^m, y)$, we obtain the desired output

$$fg = 4 + 2y + 3y^3 + 4y^4 + 3y^5 + y^6$$

Schönhage–Strassen multiplication (4/7)

- By construction, F and G both have y-degree less than m, so FG has y-degree less than $2m$
- We may thus work with $(S[y]/\langle y^{2m}+1\rangle)[x]$ in place of $S[x,y]$ when computing FG from given F and G
- Accordingly, let $R = S[y]/\langle y^{2m}+1\rangle$
- Restating our goal from the previous slide, given $F, G \in R[x]$ as input, we seek to compute a $H \in R[x]$ of x-degree at most $t-1$ such that there is a $Q \in R[x]$ with $FG = (x^t+1)Q + H$

Schönhage–Strassen multiplication (5/7)

- Next we want to reduce our goal from multiplying modulo $x^t + 1$ to multiplying modulo $x^t - 1$, since the latter can be implemented with cyclic convolution
- Toward this end, it will be useful to have a primitive root of unity of order $2t$ in R; here is where our foresight in the choice of the parameters m and t will pay off
- First, observe that y is a primitive root of unity of order $4m$ in $R = S[y]/\langle y^{2m} + 1\rangle$: indeed, since $y^{2m} \equiv -1 \pmod{y^{2m} + 1}$ holds and 2 is a unit in S by assumption, we observe that $y^{2m} - 1 \equiv -2 \pmod{y^{2m} + 1}$ is a unit in R and hence cannot be a zero divisor in R
- Since m and t are positive integer powers of 2 with $t \leq 2m$, we have that
$$\eta = y^{2m/t}$$
is a primitive root of order $2t$ in R by Lemma 2

Running example (4/8)

- Continuing the running example, we have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$
- Accordingly, in $R = S[y]/\langle y^{2m} + 1\rangle$ we have that $\eta = y^{2m/t} = y$ is a primitive root of order $2t$
- Indeed, in R we have

$$\eta^0 = 1,\ \eta^1 = y,\ \eta^2 = y^2,\ \eta^3 = y^3,\ \eta^4 = -1,\ \eta^5 = -y,\ \eta^6 = -y^2,\ \eta^7 = -y^3,\ \eta^8 = 1$$

Schönhage–Strassen multiplication (6/7)

- Given $F, G \in R[x]$ as input, we seek to compute a $H \in R[x]$ of x-degree at most $t-1$ such that there is a $Q \in R[x]$ with

$$FG = (x^t + 1)Q + H \tag{13}$$

- Observing that $\eta^t = -1$ in R and substituting ηx in place of x in (13), we have, in $R[x]$,

$$\begin{aligned} F(\eta x)G(\eta x) &= ((\eta x)^t + 1)Q(\eta x) + H(\eta x) \\ &= (-x^t + 1)Q(\eta x) + H(\eta x) \\ &= (x^t - 1)\tilde{Q}(\eta x) + H(\eta x) \end{aligned}$$

- That is, we have $F(\eta x)G(\eta x) = H(\eta x)$ in $R[x]/\langle x^t - 1\rangle$
- In particular, we can use cyclic convolution and the FFT at the primitive root of unity $\omega = \eta^2$ of order t in R to multiply $F(\eta x)$ and $G(\eta x)$ in $R[x]/\langle x^t - 1\rangle$ to obtain $H(\eta x)$
- Substituting $\eta^{-1}x$ in place of x in $H(\eta x)$ yields our desired result H in $R[x]$

Running example (5/8)

- Continuing the running example, we have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$ and

$$\begin{aligned} F &= 1 + 2y + (2 + 4y)x + (3 + 4y)x^2 + (2 + 3y)x^3 \\ G &= 3 + 2y + 4x + (1 + 4y)x^2 + (1 + 2y)x^3 \\ H &= 2y + (3y + 3y^2)x + (1 + 3y)x^2 + (1 + y^2)x^3 \end{aligned}$$

- Recalling that $\eta = y$ in $R = S[y]/\langle y^{2m} + 1\rangle$ for our chosen parameters, in $R[x]$ we have

$$\begin{aligned} F(\eta x) &= 1 + 2y + (2y + 4y^2)x + (3y^2 + 4y^3)x^2 + (2 + 2y^3)x^3 \\ G(\eta x) &= 3 + 2y + 4yx + (y^2 + 4y^3)x^2 + (3 + y^3)x^3 \\ H(\eta x) &= 2y + (3y^2 + 3y^3)x + (y^2 + 3y^3)x^2 + (4y + y^3)x^3 \end{aligned}$$

- In particular, observe how substituting ηx in place of x in F, G, H cyclically shifts the coefficients (polynomials in y) with negative wrapping because $y^{2m} = -1$ in R

Schönhage–Strassen multiplication (7/7)

- Let us now summarize the algorithm in one slide

1. To multiply $f, g \in S[y]/\langle y^{mt}+1\rangle$, construct $F, G \in R[x]$ with $R = S[y]/\langle y^{2m}+1\rangle$ from f and g by introducing a new indeterminate x and substituting $y^m = x$
2. Let $\eta = y^{2m/t} \in R$ and substitute ηx in place of x to obtain $F(\eta x), G(\eta x) \in R[x]$
3. Compute the product $F(\eta x)G(\eta x) = H(\eta x) \in R[x]/\langle x^t - 1\rangle$ via cyclic convolution

$$H(\eta x) = \frac{1}{t}\,\mathrm{DFT}_{\omega^{-1}}\Big(\mathrm{DFT}_{\omega}(F(\eta x)) \cdot \mathrm{DFT}_{\omega}(G(\eta x))\Big)$$

   using t-point fast Fourier transforms at the primitive root $\omega = \eta^2$ of order t in R

   [[This leads to t recursive multiplications in $R = S[y]/\langle y^{2m}+1\rangle$ when taking the pointwise product $\cdot$ above.]]
4. Substitute $\eta^{-1}x$ in place of x in $H(\eta x)$ to obtain H
5. Substitute $x = y^m$ and $y^{mt} = -1$ in H to obtain the output $fg \in S[y]/\langle y^{mt}+1\rangle$

Running example (6/8)

- Let us illustrate the execution of the algorithm in our running example
- We have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$ and the input

$$\begin{aligned} f &= 1 + 2y + 2y^2 + 4y^3 + 3y^4 + 4y^5 + 2y^6 + 3y^7 \\ g &= 3 + 2y + 4y^2 + y^4 + 4y^5 + y^6 + 2y^7 \end{aligned}$$

1. Substituting $y^m = x$, we construct the polynomials

$$\begin{aligned} F &= 1 + 2y + (2 + 4y)x + (3 + 4y)x^2 + (2 + 3y)x^3 \\ G &= 3 + 2y + 4x + (1 + 4y)x^2 + (1 + 2y)x^3 \end{aligned}$$

2. Substituting ηx in place of x, we obtain

$$\begin{aligned} F(\eta x) &= 1 + 2y + (2y + 4y^2)x + (3y^2 + 4y^3)x^2 + (2 + 2y^3)x^3 \\ G(\eta x) &= 3 + 2y + 4yx + (y^2 + 4y^3)x^2 + (3 + y^3)x^3 \end{aligned}$$

Running example (7/8)

- We have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$

3. Taking the t-point fast Fourier transforms at $\omega = \eta^2$, we obtain

$$\mathrm{DFT}_\omega(F(\eta x)) = (3 + 4y + 2y^2 + y^3, 2 + 4y + 3y^3, 4 + 4y^2 + 2y^3, 4y^2 + 4y^3)$$
$$\mathrm{DFT}_\omega(G(\eta x)) = (1 + y + y^2, 3 + 3y + y^2, 3y + y^2 + 3y^3, 3 + y + 2y^2 + 2y^3)$$

This leads to t recursive multiplications in $R = S/\langle y^{2m} + 1\rangle$ as follows

$$(3 + 4y + 2y^2 + y^3)(1 + y + y^2) = y + 4y^2 + 2y^3$$
$$(2 + 4y + 3y^3)(3 + 3y + y^2) = 2 + 4y^2 + 3y^3$$
$$(4 + 4y^2 + 2y^3)(3y + y^2 + 3y^3) = 3y + 3y^2 + 4y^3$$
$$(4y^2 + 4y^3)(3 + y + 2y^2 + 2y^3) = 3 + 4y + 4y^2 + y^3$$

That is, we obtain the pointwise product

$$\mathrm{DFT}_\omega(F(\eta x))\cdot\mathrm{DFT}_\omega(G(\eta x)) = (y+4y^2+2y^3, 2+4y^2+3y^3, 3y+3y^2+4y^3, 3+4y+4y^2+y^3)$$

Running example (8/8)

- We have $S = \mathbb{Z}_5$, $n = 8$, $m = 2$, $t = 4$

3. (continued)
   Taking the t-point inverse FFT, we obtain
   $$\frac{1}{t}\,\mathrm{DFT}_{\omega^{-1}}(\mathrm{DFT}_\omega(F(\eta x)) \cdot \mathrm{DFT}_\omega(G(\eta x))) = (2y, 3y^2 + 3y^3, y^2 + 3y^3, 4y + y^3)$$
   or what is the same as a polynomial in two variables
   $$H(\eta x) = 2y + (3y^2 + 3y^3)x + (y^2 + 3y^3)x^2 + (4y + y^3)x^3$$
4. Substituting $\eta^{-1}x$ in place of x in $H(\eta x)$, we obtain
   $$H = 2y + (3y + 3y^2)x + (1 + 3y)x^2 + (1 + y^2)x^3$$
5. Finally, we substitute $x = y^m$ and $y^{mt} - 1$ in H to obtain the output
   $$fg = 4 + 2y + 3y^3 + 4y^4 + 3y^5 + y^6$$

Implementation remarks (1/3)

- In an implementation, we can represent a polynomial $f \in S[y]/\langle y^{mt} + 1\rangle$ as an array consisting of mt elements of S
- Accordingly, we can represent a polynomial $F \in (S[y]/\langle y^{2m} + 1\rangle)[x]/\langle x^t - 1\rangle$ as an array of length $2mt$ that has been (tacitly) partitioned into t segments, with each segment consisting of $2m$ elements of S
- That is, each segment represents a coefficient in $R = S[y]/\langle y^{2m} + 1\rangle$ and the t segments together represent a polynomial in $R[x]/\langle x^t - 1\rangle$

Implementation remarks (2/3)

- Multiplication with powers of y in $R = S[y]/\langle y^{2m} + 1\rangle$ is easy: we just cyclically shift the list of coefficients of a polynomial in y by as many places as is indicated by the power of y, taking care to adjust the sign of the coefficient in case of wrap-arounds
- Accordingly, multiplication and substitution with powers of η are similarly negative-wrapping cyclic shifts
- In particular, fast Fourier transforms at $\omega = \eta^2$ over $R = S[y]/\langle y^{2m} + 1\rangle$ similarly amount to additions and negative-wrapping cyclic shifts

Implementation remarks (3/3)

- To build F from f, we (i) view f as a collection of t segments of length m each, and (ii) pad each segment with m zeros of S so that each segment has length $2m$
- Multiplication and substitution with a power of η rotates each segment cyclically (with negative wrapping since $y^{2m} = -1$)
- Recursive multiplications in R operate on pairs of segments
- To build fg from H, we compress back from length $2mt$ to length mt so that each of the mt elements of fg becomes a (signed) sum of 2 elements of H as determined by the substitutions $x = y^m$ and $y^{tm} = -1$

Analysis (1/3)

- For $n = 2^k$ with $k \in \mathbb{Z}_{\geq 0}$, we claim that Schönhage-Strassen multiplication runs in $O(n \log n \log \log n)$ operations in S for two inputs $f, g \in S[y]/\langle y^n + 1 \rangle$ given in coefficient representation
- Recalling that $t = 2^{\lceil k/2 \rceil}$ and $m = 2^{\lfloor k/2 \rfloor}$ with $n = mt \geq 8$, it suffices to analyse the recurrence

$$T(n) \leq tT(2m) + Cn \log_2 n \tag{14}$$

  with $T(1), T(2), T(4) \leq D$ where C and D are constants independent of n
- Indeed, for an input of size $n \geq 8$, the algorithm makes t recursive calls on inputs of size $2m < n$ and does at most $Cn \log_2 n$ work (operations in S) to prepare the recursive calls and to prepare the result based on the return values of the calls

Analysis (2/3)

- Let us reparameterize (14) in terms of k to obtain, for all $k \geq 3$,

$$T(k) \leq 2^{\lceil k/2 \rceil} T(\lfloor k/2 \rfloor + 1) + C \cdot 2^k k \tag{15}$$

- For all nonnegative integers k we have

$$\lfloor k/2 \rfloor + \lceil k/2 \rceil = k, \quad \lfloor (k+1)/2 \rfloor = \lceil k/2 \rceil, \quad \text{and} \quad \lceil (k+1)/2 \rceil = \lfloor k/2 \rfloor + 1 \tag{16}$$

- From (16) we have that (15) is equivalent to, for all $k \geq 2$,

$$T(k+1) \leq 2^{\lfloor k/2 \rfloor + 1} T(\lceil k/2 \rceil + 1) + C \cdot 2^{k+1}(k+1) \tag{17}$$

- For convenience, let us substitute $T(k+1) = 2^k (k-1) L(k)$ to (17) and divide by $2^k(k-1)$ on both sides to obtain the equivalent form, for all $k \geq 2$,

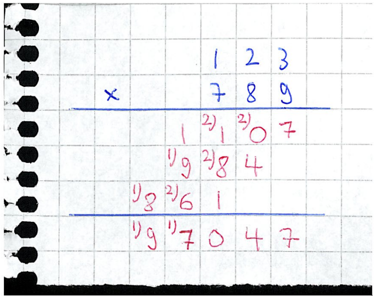
$$L(k) \leq \frac{2(\lceil k/2 \rceil - 1)}{k-1} L(\lceil k/2 \rceil) + \frac{2C(k+1)}{k-1} \tag{18}$$

Analysis (3/3)

- From (18) we obtain that for all $k \geq 2$ we have
$$L(k) \leq L(\lceil k/2 \rceil) + 6C \tag{19}$$
- Now let us observe that at most $\log_2 3k$ iterations of the map $k \mapsto \lceil k/2 \rceil$ suffice to reach the value 1 starting from any positive integer k
- Indeed, the map $k \mapsto \lceil k/2 \rceil$ is dominated by the map $k \mapsto \lfloor k/2 \rfloor + 1$, which can be viewed as right-shifting k (viewed as an integer in base 2 representation) by one bit position and then incrementing the result
- When iterating $k \mapsto \lfloor k/2 \rfloor + 1$, the increments in total contribute at most the least power of 2 at least k (which is at most $2k$), so at most $\log_2 3k$ right-shifts and increments suffice to reach the value 1
- In particular, iterating (19), we obtain, for all $k \geq 2$,
$$L(k) \leq L(\lceil k/2 \rceil) + 6C \leq L(\lceil \lceil k/2 \rceil /2 \rceil) + 12C \leq \cdots \leq L(1) + 6C \log_2 3k = O(\log k)$$
and for $k \geq 3$ thus $T(k) = 2^{k-1}(k-2)L(k-1) = O(2^k k \log k)$

Application

- The classical integer multiplication algorithm has quadratic complexity
- But how to handle, say, gigabyte-size operands?
- We study the key ideas for FFT-based fast integer multiplication (exercise)
- See also:
  - https://gmplib.org
  - M. Fürer, Faster integer multiplication, *SIAM J. Comput.* 39 (2009) 979–1005 [9].
  - D. Harvey, J. van der Hoeven, G. Lecerf, Even faster integer multiplication, *J. Complexity* 36 (2016) 1–30 [13].



Further remarks

- Using Schönhage–Strassen multiplication, we can multiply two polynomials of degree at most n in $O(n \log n \log\log n)$ operations in the coefficient ring S, provided that 2 is a unit in S
- With some extra work, the assumption that 2 is a unit in S can be lifted to obtain a multiplication algorithm that works over any coefficient ring S in $O(n \log n \log\log n)$ operations; cf. von zur Gathen and Gerhard [11, Section 8.3, Exercises 8.29 and 8.30] and Schönhage [23]
- Cf. also the “three-primes” FFT algorithm for integer multiplication on 64-bit hardware [11, Section 8.3]

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic, approximation, online, and randomised algorithms
- Ways of coping with uncertainty in computation, including error-correction and proofs of correctness
- The art of solving a large problem by reduction to one or more smaller instances of the same or a related problem
- (Linear) independence, dependence, and their abstractions as enablers of efficient algorithms

Learning objectives (2/2)

- Making use of duality
  - Often a problem has a corresponding **dual** problem that is obtainable from the original (the **primal**) problem by means of an easy transformation
  - The primal and dual control each other, enabling an algorithm designer to use the interplay between the two representations
- Relaxation and tradeoffs between objectives and resources as design tools
  - Instead of computing the exact optimum solution at considerable cost, often a less costly but principled approximation suffices
  - Instead of the complete dual, often only a randomly chosen partial dual or other relaxation suffices to arrive at a solution with high probability

Recap of key content for Lecture 2

- Evaluation–interpolation duality of polynomials
- Multiplication is a pointwise product in the dual
- Transforming between the primal and a (carefully chosen) dual —**roots of unity** and the **discrete Fourier transform** (DFT)
- The positional number system for integers
- Factoring a composite-order DFT to obtain a **fast Fourier transform** (FFT)
- Fast cyclic convolution (assuming a suitable root of unity exists)
- Fast negative-wrapping cyclic convolution